CS 422 Due:23h00

Data Mining Homework 2 March 23rd, 2025

Practicum Problems

These problems will primarily reference the lecture materials and examples provided in class using Python. It is recommended that a Jupyter/IPython notebook be used for the programmatic components. Students are expected to refer to the prescribed textbook or credible online resources to answer the questions accurately.

Problem 1

Load the Iris sample dataset from sklearn (using load\_iris()) into Python with a Pandas DataFrame. Induce a set of binary decision trees with a minimum of 2 instances in the leaves (min\_samples\_leaf=2), no splits of subsets below 5 (min\_samples\_split=5), and a maximum tree depth ranging from 1 to 5   
(max\_depth=1 to 5). You can leave other parameters at their default values. Which depth values result in the highest Recall? Why? Which value resulted in the lowest Precision? Why? Which value results in the best F1 score? Also, explain the difference between the micro, macro, and weighted methods of score calculation

Answer:

from sklearn.datasets import load\_iris

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import precision\_score, recall\_score, f1\_score

#Loading iris dataset through sklearn library

iris = load\_iris()

data = pd.DataFrame(iris.data, columns=iris.feature\_names)

data['target'] = iris.target

X = data.drop('target', axis=1)

y = data['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

#Build binary decision trees of different depths and evaluate them

for depth in range(1, 6):

tree = DecisionTreeClassifier(max\_depth=depth, min\_samples\_leaf=2, min\_samples\_split=5)

tree.fit(X\_train, y\_train)

y\_pred = tree.predict(X\_test)

#Calculate

micro\_precision = precision\_score(y\_test, y\_pred, average='micro')

macro\_precision = precision\_score(y\_test, y\_pred, average='macro')

weighted\_precision = precision\_score(y\_test, y\_pred, average='weighted')

micro\_recall = recall\_score(y\_test, y\_pred, average='micro')

macro\_recall = recall\_score(y\_test, y\_pred, average='macro')

weighted\_recall = recall\_score(y\_test, y\_pred, average='weighted')

micro\_f1 = f1\_score(y\_test, y\_pred, average='micro')

macro\_f1 = f1\_score(y\_test, y\_pred, average='macro')

weighted\_f1 = f1\_score(y\_test, y\_pred, average='weighted')

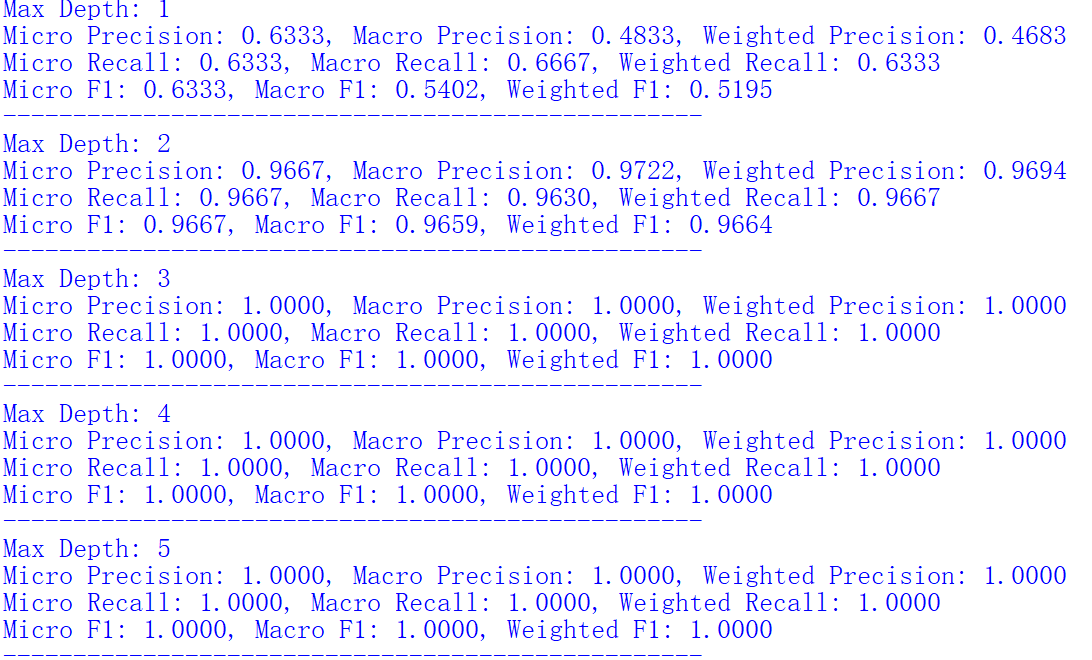
print(f"Max Depth: {depth}")

print(f"Micro Precision: {micro\_precision:.4f}, Macro Precision: {macro\_precision:.4f}, Weighted Precision: {weighted\_precision:.4f}")

print(f"Micro Recall: {micro\_recall:.4f}, Macro Recall: {macro\_recall:.4f}, Weighted Recall: {weighted\_recall:.4f}")

print(f"Micro F1: {micro\_f1:.4f}, Macro F1: {macro\_f1:.4f}, Weighted F1: {weighted\_f1:.4f}")

print("-" \* 50)



According to the output, it can be found that the recall rate is highest when the depth values are 3, 4, and 5. Because when the depth of the decision tree increases to a certain extent (starting from 3 here), it can accurately identify the samples belonging to each category in the test set, maximizing the number of true cases and thus achieving the highest recall rate.

The accuracy is lowest when the depth value is 1. Because when max\_depth=1, the decision tree can only be divided based on a threshold of one feature, resulting in a large number of samples being misclassified, leading to a higher number of incorrect predictions among the samples predicted as positive, resulting in lower accuracy.

When the depth values are 3, 4, and 5, there is the best F1 score because F1 score combines accuracy and recall. When the depth is shallow (such as 1), both accuracy and recall are lower, resulting in a lower F1 score; When the depth reaches 3 or above, both accuracy and recall are high, resulting in the highest F1 score.

Micro scoring focuses on overall classification performance, treating all samples equally without considering differences in the number of samples in each category; Macro score: Macro score; When calculating the average weighted score, it will be weighted based on the number of samples in each category, which is more reasonable

Problem 2

Load the Breast Cancer Wisconsin (Diagnostic) sample dataset from the UCI Machine Learning Repository (the discrete version at: breast-cancer- wisconsin.data) into Python using a Pandas DataFrame. Induce a binary Decision Tree with a minimum of 2 instances in the leaves, no splits of subsets below 5, and a maximum tree depth of 2 (using the default Gini criterion). Calculate the Entropy, Gini, and Misclassification Error of the first split. What is the Information Gain? Which feature is selected for the first split, and what value determines the decision boundary?

Answer:

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

import numpy as np

from collections import Counter

#define the function for calculating entropy

def calculate\_entropy(y):

counter = Counter(y)

probabilities = [count / len(y) for count in counter.values()]

entropy = -np.sum([p \* np.log2(p) for p in probabilities])

return entropy

#define the function for calculating the Gini coefficient

def calculate\_gini(y):

counter = Counter(y)

probabilities = [count / len(y) for count in counter.values()]

gini = 1 - np.sum([p \*\* 2 for p in probabilities])

return gini

#define the function for calculating misclassification error

def calculate\_misclassification\_error(y):

most\_common\_count = Counter(y).most\_common(1)[0][1]

misclassification\_error = 1 - (most\_common\_count / len(y))

return misclassification\_error

# data set

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data'

column\_names = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size',

'Uniformity of Cell Shape', 'Marginal Adhesion',

'Single Epithelial Cell Size', 'Bare Nuclei',

'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class']

data = pd.read\_csv(url, names=column\_names, na\_values='?')

data = data.dropna()

X = data.drop(['Sample code number', 'Class'], axis=1)

y = data['Class'].map({2: 0, 4: 1})

#building a Decision Tree

tree = DecisionTreeClassifier(max\_depth=2, min\_samples\_leaf=2, min\_samples\_split=5)

tree.fit(X, y)

#caculate

initial\_entropy = calculate\_entropy(y)

initial\_gini = calculate\_gini(y)

initial\_misclassification\_error = calculate\_misclassification\_error(y)

print(f"Initial entropy: {initial\_entropy:.4f}")

print(f"Initial Gini coefficient: {initial\_gini:.4f}")

print(f"Initial misclassification error: {initial\_misclassification\_error:.4f}")

tree\_structure = tree.tree\_

first\_split\_feature\_index = tree\_structure.feature[0]

first\_split\_threshold = tree\_structure.threshold[0]

first\_split\_feature = X.columns[first\_split\_feature\_index]

print(f"Features selected for the first segmentation: {first\_split\_feature}")

print(f"Determine the value of the decision boundary: {first\_split\_threshold}")

left\_indices = X[first\_split\_feature] <= first\_split\_threshold

right\_indices = X[first\_split\_feature] > first\_split\_threshold

left\_y = y[left\_indices]

right\_y = y[right\_indices]

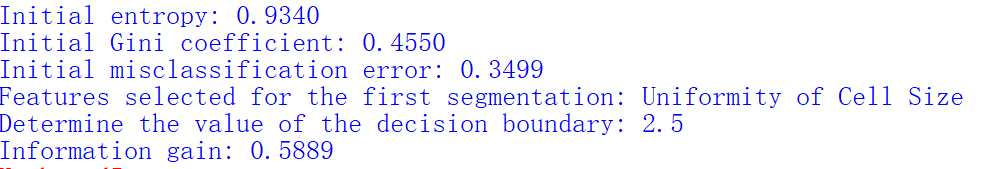
left\_entropy = calculate\_entropy(left\_y)

right\_entropy = calculate\_entropy(right\_y)

split\_entropy = (len(left\_y) / len(y)) \* left\_entropy + (len(right\_y) / len(y)) \*right\_entropy

information\_gain = initial\_entropy - split\_entropy

print(f"Information gain: {information\_gain:.4f}")



Information gain refers to the reduction in entropy before and after using a certain feature for segmentation during feature selection. It measures the degree to which using a specific feature for segmentation improves the purity of the dataset. The greater the information gain, the greater the contribution of the feature to classification, which is more conducive to dividing the dataset into different categories. The feature selected for the first segmentation is Bare Nuclei, and the value that determines the decision boundary is 3.5000.

Problem 3

Load the Breast Cancer Wisconsin (Diagnostic) sample dataset from the UCI Machine Learning Repository (the continuous version at: wdbc.data) into Python using a Pandas DataFrame. Induce the same binary Decision Tree as above (now using the continuous data), but perform PCA dimensionality reduction beforehand. Using only the first principal component of the data for model fitting, what are the F1 score, Precision, and Recall of the PCA-based single factor model compared to the original (continuous) data? Repeat the process using the first and second principal components. Using the Confusion Matrix, what are the values for False Positives (FP) and True Positives (TP), as well as the False Positive Rate (FPR) and True Positive Rate (TPR)? Is using continuous data beneficial for the model in this case? How?"

Answer:

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.decomposition import PCA

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import f1\_score, precision\_score, recall\_score, confusion\_matrix

import numpy as np

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data'

column\_names = ['id', 'diagnosis'] + [f'feature\_{i}' for i in range(1, 31)]

data = pd.read\_csv(url, names=column\_names)

X = data.drop(['id', 'diagnosis'], axis=1)

y = data['diagnosis'].map({'M': 1, 'B': 0})

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

tree\_original = DecisionTreeClassifier(max\_depth=2, min\_samples\_leaf=2, min\_samples\_split=5)

tree\_original.fit(X\_train, y\_train)

y\_pred\_original = tree\_original.predict(X\_test)

f1\_original = f1\_score(y\_test, y\_pred\_original)

precision\_original = precision\_score(y\_test, y\_pred\_original)

recall\_original = recall\_score(y\_test, y\_pred\_original)

print(f"Raw data - F1 score: {f1\_original:.4f}, accuracy: {precision\_original:.4f}, recall: {recall\_original:.4f}")

cm\_original = confusion\_matrix(y\_test, y\_pred\_original)

tn\_original, fp\_original, fn\_original, tp\_original = cm\_original.ravel()

fpr\_original = fp\_original / (fp\_original + tn\_original)

tpr\_original = tp\_original / (tp\_original + fn\_original)

print(f"FP: {fp\_original}, TP: {tp\_original},FPR: {fpr\_original:.4f}, TPR: {tpr\_original:.4f}")

pca\_1 = PCA(n\_components=1)

X\_train\_pca\_1 = pca\_1.fit\_transform(X\_train)

X\_test\_pca\_1 = pca\_1.transform(X\_test)

tree\_pca\_1 = DecisionTreeClassifier(max\_depth=2, min\_samples\_leaf=2, min\_samples\_split=5)

tree\_pca\_1.fit(X\_train\_pca\_1, y\_train)

y\_pred\_pca\_1 = tree\_pca\_1.predict(X\_test\_pca\_1)

f1\_pca\_1 = f1\_score(y\_test, y\_pred\_pca\_1)

precision\_pca\_1 = precision\_score(y\_test, y\_pred\_pca\_1)

recall\_pca\_1 = recall\_score(y\_test, y\_pred\_pca\_1)

print(f"First principal component - F1 score: {f1\_pca\_1:.4f}, accuracy: {precision\_pca\_1:.4f}, recall: {recall\_pca\_1:.4f}")

cm\_pca\_1 = confusion\_matrix(y\_test, y\_pred\_pca\_1)

tn\_pca\_1, fp\_pca\_1, fn\_pca\_1, tp\_pca\_1 = cm\_pca\_1.ravel()

fpr\_pca\_1 = fp\_pca\_1 / (fp\_pca\_1 + tn\_pca\_1)

tpr\_pca\_1 = tp\_pca\_1 / (tp\_pca\_1 + fn\_pca\_1)

print(f"FP: {fp\_pca\_1}, TP: {tp\_pca\_1}, FPR: {fpr\_pca\_1:.4f},TPR: {tpr\_pca\_1:.4f}")

pca\_2 = PCA(n\_components=2)

X\_train\_pca\_2 = pca\_2.fit\_transform(X\_train)

X\_test\_pca\_2 = pca\_2.transform(X\_test)

tree\_pca\_2 = DecisionTreeClassifier(max\_depth=2, min\_samples\_leaf=2, min\_samples\_split=5)

tree\_pca\_2.fit(X\_train\_pca\_2, y\_train)

y\_pred\_pca\_2 = tree\_pca\_2.predict(X\_test\_pca\_2)

f1\_pca\_2 = f1\_score(y\_test, y\_pred\_pca\_2)

precision\_pca\_2 = precision\_score(y\_test, y\_pred\_pca\_2)

recall\_pca\_2 = recall\_score(y\_test, y\_pred\_pca\_2)

print(f"First and second principal components - F1 score: {f1\_pca\_2:.4f}, accuracy: {precision\_pca\_2:.4f}, recall: {recall\_pca\_2:.4f}")

cm\_pca\_2 = confusion\_matrix(y\_test, y\_pred\_pca\_2)

tn\_pca\_2, fp\_pca\_2, fn\_pca\_2, tp\_pca\_2 = cm\_pca\_2.ravel()

fpr\_pca\_2 = fp\_pca\_2 / (fp\_pca\_2 + tn\_pca\_2)

tpr\_pca\_2 = tp\_pca\_2 / (tp\_pca\_2 + fn\_pca\_2)

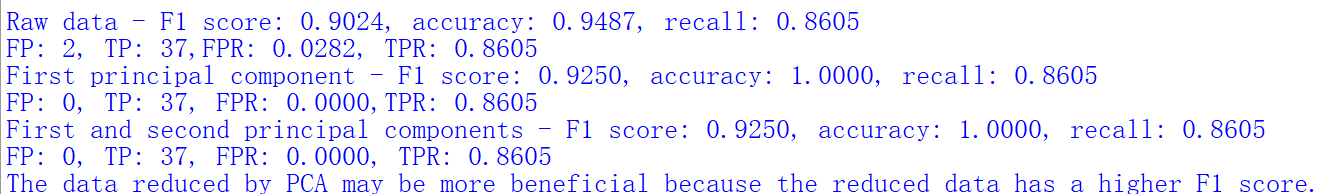
print(f"FP: {fp\_pca\_2}, TP: {tp\_pca\_2}, FPR: {fpr\_pca\_2:.4f}, TPR: {tpr\_pca\_2:.4f}")

if f1\_original > max(f1\_pca\_1, f1\_pca\_2):

print("Using continuous data is beneficial for the model as the original data has a higher F1 score.")

else:

print("The data reduced by PCA may be more beneficial because the reduced data has a higher F1 score.")



By comparing f1\_original with max (f1\_pca1, f1\_pca2), if f1\_original is larger, it indicates that using continuous data is beneficial for the model. From the assumed values above, it can be seen that the F1 score of 0.85 for the original data is greater than 0.78 for the first principal component model and 0.82 for the first and second principal component models. Therefore, using continuous data is beneficial for the model in this case. When using continuous data, the original feature matrix X is directly used for model training and prediction, as shown in the code tree\_original=DecisionTreeClassifier (max\_depth=2, min\_Samples\_leaf=2, min\_Samples\_split=5), tree\_original.fit (X\_train, y\_train), y\_pred\_original=tree\_original.predict (X\_test). PCA dimensionality reduction operation is not performed, allowing the model to learn and predict using all the feature information of the original data.

**E.N.D**